MC-102 — Aula 06
Comandos de Repetição I: `while` e `do-while`

Instituto de Computação – Unicamp

Segundo Semestre de 2011

Roteiro

1. Simulação de código
2. Comando de Repetição
3. while (condicao) { comandos }
4. do { comandos } while (condicao);
5. Exemplos

Introdução

- Às vezes, acontece de programarmos um código, porém ele não faz o que esperávamos que fizesse.
- Isso acontece por vários motivos, entre os quais destacam-se:
  - Erros de programação: instruções escritas erradas.
  - Erros da nossa lógica: o conjunto de passos pensados que parecia resolver o problema na realidade não cobre todas as situações.
- Eventualmente, simplesmente olhar o código pode não trazer a tona o erro.
- Por isso, utiliza-se uma técnica de simulação do código
  - Pode ser automatizada (utilizando um *debugger*)
  - Pode ser feita manualmente, utilizando papel e caneta.

Simulação Manual

- Bem simples: Existem apenas 2 passos.
  - "Alocação" dos espaços de variáveis
  - "Execução" de uma instrução de cada vez.
- Alocação de memória:
  - Ex. Suponha o código:
    1. `int divisor,dividendo;`
    2. `float resultado;`
- Após "executar" a linha 1

Tipo	`int`	`int`
Nome	`divisor`	`dividendo`
Valor	?	?

Simulação Manual

- Bem simples: Existem apenas 2 passos.
  - "Alocação" dos espaços de variáveis
  - "Execução" de uma instrução de cada vez.
- Alocação de memória:
  - Ex. Suponha o código:
    1. `int divisor,dividendo;`
    2. `float resultado;`
- Após "executar" a linha 2

Tipo	`int`	`int`	`float`
Nome	`divisor`	`dividendo`	`resultado`
Valor	?	?	?

Simulação Manual

- Execução em memória:
  - Ex. Suponha o código:
    1. `int divisor,dividendo;`
    2. `float resultado;` ← Último executado
    3. `divisor=10;` ← Próximo Comando
    4. `dividendo=13;`
    5. `resultado = dividendo / divisor;`
- Após "executar" a linha 2

Tipo	`int`	`int`	`float`
Nome	`divisor`	`dividendo`	`resultado`
Valor	?	?	?

Simulação Manual

- Execução em memória:
  - Ex. Suponha o código:
    1. `int divisor,dividendo;`
    2. `float resultado;`
    3. `divisor=10;` ← Último executado
    4. `dividendo=13;` ← Próximo Comando
    5. `resultado = dividendo / divisor;`
- Após "executar" a linha 3

Tipo	`int`	`int`	`float`
Nome	`divisor`	`dividendo`	`resultado`
Valor	? 10	?	?

Simulação Manual

- Execução em memória:
  - Ex. Suponha o código:
    1. `int divisor,dividendo;`
    2. `float resultado;`
    3. `divisor=10;`
    4. `dividendo=13;` ← Último executado
    5. `resultado = dividendo / divisor;` ← Próximo Comando
- Após "executar" a linha 4

Tipo	`int`	`int`	`float`
Nome	`divisor`	`dividendo`	`resultado`
Valor	? 10	? 13	?

Simulação Manual

- Execução em memória:
  - Ex. Suponha o código:
    1. `int divisor,dividendo;`
    2. `float resultado;`
    3. `divisor=10;`
    4. `dividendo=13;`
    5. `resultado = dividendo / divisor;` ← Último executado
- Após "executar" a linha 5

Tipo	`int`	`int`	`float`
Nome	`divisor`	`dividendo`	`resultado`
Valor	? 10	? 13	? 1.0

Simulação Manual

- Execução em memória:
  - Ex. Suponha o código:
    1. `int divisor,dividendo;`
    2. `float resultado;`
    3. `divisor=10;`
    4. `dividendo=13;`
    5. `resultado = dividendo / divisor;` ← Último executado
- Término da execução (não há mais comandos)

Tipo	`int`	`int`	`float`
Nome	`divisor`	`dividendo`	`resultado`
Valor	? 10	? 13	? 1.0

Simulação Manual

- Execução em memória:
  - Ex. Suponha o código (corrigido):
    1. `int divisor,dividendo;`
    2. `float resultado;`
    3. `divisor=10;`
    4. `dividendo=13;`
    5. `resultado = (float)dividendo / (float)divisor;`
- Execução completa

Tipo	`int`	`int`	`float`
Nome	`divisor`	`dividendo`	`resultado`
Valor	?	?	?
	10	13	1.3

Introdução

- Até agora, vimos como escrever programas capazes de executar comandos de forma linear, e, se necessário, tomar decisões com relação a executar ou não um bloco de comandos.
- Entretanto, eventualmente é necessário executar um bloco de comandos várias vezes para obter o resultado.

Exemplo

Calcule a divisão inteira de dois numeros usando apenas soma e subtração

Solução

- Duas variáveis: temporario, contador
  1. temporario=dividendo;
  2. contador=0;
  3. Enquanto *temporario* $>$ *divisor*
     1. temporario = temporario - divisor
     2. contador = contador + 1
  4. Exiba contador

Por que?

Contador equivale a divisão inteira de dividendo por divisor

Introdução

- Será que dá pra fazer com o que já temos?
- Ex.: Programa que imprime todos os números de 1 a 4

```
printf("1");
printf("2");
printf("3");
printf("4");
```

Introdução

- Ex.: Programa que imprime todos os números de 1 a 100

```
printf("1");
printf("2");
printf("3");
printf("4");
/*repete 95 vezes a linha acima*/
printf("100");
```

Introdução

- Ex.: Programa que imprime todos os números de 1 a n (dado)

```
printf("1");
if (n>=2)
   printf("2");
if (n>=3)
   printf("3");
/*repete 96 vezes o bloco acima*/
if (n>=100)
   printf("100");
```

Introdução

- Ex.: Programa que imprime 2 elevado a todos os números de 1 a n (dado)

```
printf("2^1 = 2");
if (n>=2)
   printf("2^2 = 4");
/*repete 97 vezes o bloco acima*/
if (n>=100)
   printf("2^100 = ???");
```

Introdução

- Ex.: Programa que imprime 2 elevado a todos os números de 1 a n (dado)

```
int i=1, pot=2;
printf("2^%d = %d",i,pot);
i++; pot *=2;
if (n>=2) {
   printf("2^%d = %d",i,pot);
   i++; pot *=2; }
/*repete 97 vezes o bloco acima*/
if (n>=100) {
   printf("2^%d = %d",i,pot);
   i++; pot *=2; }
```

Introdução

- Ex.: Programa que imprime 2 elevado a todos os números de 1 a n (dado)

```
int i=1,pot=2;
if (i<=n) {
   printf("2^%d = %d",i,pot);
   i++; pot *=2; }
/*repete 98 vezes o bloco acima*/
if (i<=n) {
   printf("2^%d = %d",i,pot);
   i++; pot *=2; }
```

Introdução

- Reparem, no exemplo anterior, que o trecho abaixo é executado 100 vezes

```
if (i<=n)
{
   printf("2^%d = %d",i,pot);
   i++;
   pot *=2;
}
```

Introdução

- Para cada comparação, fazemos:
  1. imprimimos o expoente e sua potência.
  2. incrementamos o expoente
  3. multiplicamos a potência
- Quando i supera n, todas as demais comparações retornam falso, e não são executadas.

Problema

n é limitado ao tamanho do nosso código.

Introdução

- Seria interessante fazer com que o código repetisse a comparação e executasse o comando dentro até que a condição fosse falsa

```
/* Enquanto for verdade que i<=n, execute */
{
   printf("2^%d = %d",i,pot);
   i++; pot *=2;
}
```

while (condicao) { comandos }

- Estrutura:

```
while ( condicao ) comando;
while ( condicao ) { comandos }
```

- Enquanto a condição for verdadeira (!=0), ele executa o(s) comando(s);
  - Passo 1: Testa condição. Se a condição for verdadeira vai para o Passo 2.
    - Passo 2: Executa comandos;
    - Passo 2.1: Volta para o Passo 1

Imprimindo os 100 primeiros números inteiros

```
int i=1;
while (i<=100)
{
   printf("%d ",i);
   i++;
}
```

Imprimindo os *n* primeiros números inteiros

```
int i=1,n;
scanf("%d",&n);
while (i<=n)
{
   printf("%d ",i);
   i++;
}
```

Imprimindo as *n* primeiras potências de 2

```
int i=1,n,pot=2;
scanf("%d",&n);
while (i<=n)
{
   printf("2^%d = %d ",i,pot);
   i++;
   pot*=2;
}
```

while (condicao) { comandos }

- 1. O que acontece se a condição for falsa na primeira vez?
  `while (a!=a) a=a+1;`
- 2. O que acontece se a condição for sempre verdadeira?
  `while (a==a) a=a+1;`

while (condicao) { comandos }

- 1. O que acontece se a condição for falsa na primeira vez?
  `while (a!=a) a=a+1;`

  R: Ele nunca entra na repetição (*loop*).

- 2. O que acontece se a condição for sempre verdadeira?
  `while (a==a) a=a+1;`

  R: Ele entra na repetição e nunca sai (*loop* infinito).

while (condicao) { comandos }

- Estudando a estrutura "normal" do `while` mais a fundo.
  `while (i<=n)` ← condição de repetição
  {
  `printf("%d ",i);`
  `i++;` ← Comando de passo
  }
- O oposto (negação) da condição de repetição é conhecida como condição de parada:
  `!(i<=n)` ⇒ `i>n` é a condição de parada.

while (condicao) { comandos }

- *loop* de fim determinado

```
scanf("%d",&preco);
while (i<=n) {
   total = total + preco;
   i++;
   scanf("%d",&preco);
}
```

while (condicao) { comandos }

- *loop* de fim indeterminado

```
scanf("%d",&preco);
while (preco>0) {
   total = total + preco;
   scanf("%d",&preco);
}
```

do { comandos } while (condicao);}

- Estrutura:
  `do comando; while ( condicao );`
  `do { comandos } while ( condicao );`

- Diferença do `while`: Sempre entra na primeira vez
- Teste condicional é feito por ultimo.
- Passo 1: Executa comandos;
- Passo 2: Testa condição. Se for verdadeira vai para Passo 1.

Imprimindo os n primeiros números inteiros

```
int i, n;
i=1;
scanf("%d",&n);
do{
i++;
}while(i<=n);
printf("\n %d",i);
```

- O que acontece quando digita 0 (n=0)?

MDC(x,y)

Supondo, sem perda de generalidade, que $x \geq y$, o MDC(x, y) é definido da seguinte forma:

$$\text{MDC}(x, y) = \begin{cases} y & \text{caso } x \bmod y = 0 \\ \text{MDC}(y, x \bmod y) & \text{caso contrário} \end{cases}$$

Exercício

Complete o programa em `mdc.c`

MDC(x,y)

```
x = maior;
y = menor;
do
{
   r = x % y;
   x = y;
   y = r;
} while (r!=0);
```

- Repare que `r` só é calculado dentro do *loop*
- Veja exemplo em `mdc-completo.c`

Soma de n valores inteiros

```
soma = 0;
while (n > 0) {
  printf("número a ser somado: ");
  scanf("%d", &parcela);
  soma += parcela;
  n--;
}
printf("Soma: %d\n", soma);
```

- Veja exemplo em `soma-n.c`

Soma até 0

```
soma = 0;
printf("número a ser somado (0 para sair): ");
scanf("%d", &parcela);

while (parcela != 0) {
  soma += parcela;
  printf("número a ser somado (0 para sair): ");
  scanf("%d", &parcela);
}

printf("Soma: %d\n", soma);
```

- Veja exemplo em `soma-ate-0.c`

Soma até 0

```
soma = 0;

do {
  printf("número a ser somado (0 para sair): ");
  scanf("%d", &parcela);
  soma += parcela;
} while (parcela != 0);

printf("Soma: %d\n", soma);
```

- Veja exemplo em `soma-ate-0-do-while.c`

Arte em ASCII

Como imprimir uma linha de '*'s

****

- Veja exemplo em `linha.c`

Arte em ASCII

```
**********
*********
********
*******
******
*****
****
***
**
*
```

- Veja exemplo em `desenho.c`

Arte em ASCII

*****
****
***
**
*
*
**
***
****
*****

- Veja exemplo em `desenho2.c`

Arte em ASCII

```
**********
*****  *****
****    ****
***      ***
**        **
*          *
*          *
**        **
***      ***
****    ****
*****  *****
**********
```

- Veja exemplo em `desenho3.c`

Arte em ASCII

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

- Veja exemplo em `desenho4.c`